Theoretical Computer Science 410 (2009) 4315–4326

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Interchange rearrangement: The element-cost model

Oren Kapah [b], Gad M. Landau [a,e,1], Avivit Levy [c,d,*], Nitsan Oz [a,1]

[a] Department of Computer Science, University of Haifa, Haifa 31905, Israel

[b] Department of Computer Science, Bar Ilan University, Ramat Gan 52900, Israel

[c] Department of Software Engineering, Shenkar College, 12 Anna Frank, Ramat-Gan, 52526, Israel

[d] CRI, University of Haifa, Haifa 31905, Israel

[e] Department of Computer and Information Science, Polytechnic Institute of NYU, Six MetroTech Center, Brooklyn, NY 11201-3840, United States

ARTICLE INFO

Keywords:
Strings rearrangement distances
Rearrangement cost models
Interchange rearrangement

ABSTRACT

Given an input string S and a target string T when S is a permutation of T, the *interchange rearrangement problem* is to apply on S a sequence of interchanges, such that S is transformed into T. The *interchange* operation exchanges the position of the two elements on which it is applied. The goal is to transform S into T at the minimum cost possible, referred to as the distance between S and T. The distance can be defined by several cost models that determine the cost of every operation. There are two known models: The *Unit-cost model* and the *Length-cost model*. In this paper, we suggest a natural cost model: The *Element-cost model*. In this model, the cost of an operation is determined by the elements that participate in it. Though this model has been studied in other fields, it has never been considered in the context of rearrangement problems. We consider both the special case where all elements in S and T are distinct, referred to as a *permutation string*, and the general case, referred to as a *general string*. An efficient optimal algorithm for the *permutation string* case and efficient approximation algorithms for the *general string* case, which is $\mathcal{NP}$-hard, are presented. The study is broadened to include the *transposition rearrangement problem* under the *Element-cost model* and under the other known models, in order to provide additional perspective on the new model.

1. Introduction

The problem of defining the distance or similarity between two strings S and T has been studied extensively over the years. There are many known and established methods, such as the *Edit distance* and the *Hamming distance* [13]. The *Edit distance* allows three operations (substitution, insertion or deletion) upon the input string. There are several generalizations of the basic *Edit distance* (also referred to as the *Levenshtein distance*), which defines a unit-cost for every operation. One is the *the operation-weight edit distance*, which gives a unit-cost for every type of operation. Another is the *alphabet-weight edit distance*, which defines a cost for every operation depending on the elements participating in the specific operation.

These string metrics deal with errors of data appearing in the text and give a measure of either similarity or distance between an input string S and a target string T. The order of the elements is assumed to be correct. However, address errors may also be considered ([1–4]). In these types of errors, elements in S may only be mispositioned. It is commonly assumed that the input string is a permutation of the target string in order to have a finite distance. In the *rearrangement problem*, it

* Corresponding author at: Department of Software Engineering, Shenkar College, 12 Anna Frank, Ramat-Gan, 52526, Israel.
E-mail addresses: kapaho@cs.biu.ac.il (O. Kapah), landau@cs.haifa.ac.il (G.M. Landau), avivitlevy@shenkar.ac.il (A. Levy), nitsan@cri.haifa.ac.il (N. Oz).
1 Tel.: +972 4 824 0103; fax: +972 4 824 9331.

0304-3975/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2009.07.013

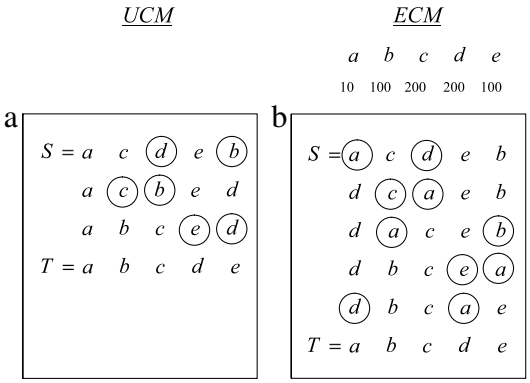

Fig. 1. In both (a) and (b), every row represents a stage in the rearrangement. The elements marked with circles are the elements interchanged to establish the next stage. In (a), the goal is to transform S into T with a minimum number of interchanges (*UCM*). This is done by applying 3 interchanges. In (b), the *ECM* is used. Every element has a weight and the cost of an interchange is the sum of the weights. The sequence of interchanges applied in (a) costs 900, whereas the sequence of 5 interchanges applied in (b) costs 850.

is assumed that only address errors have occurred. The goal is to apply a sequence of legal operations on S, such that S is transformed into T at the minimum cost possible, referred to as the distance between S and T.

The *interchange rearrangement problem* was studied by Cayley [9]. Cayley solved this problem for *permutation strings* under the *Unit-cost model* and left the problem of *general strings* as an open problem. Recently, Amir et al. solved Cayley's open problem by showing it is $\mathcal{NP}$-hard and giving a 1.5-approximation algorithm. In addition, they extended this problem by examining it under the *Length-cost model* [4]. In this paper, we further extend this problem on both *permutation strings* and *general strings* by examining it under the *Element-cost model*.

Formal definitions. We begin with formal definitions of the *interchange* operator and the *Element-cost model*.

Definition 1. Let $S = s_1, \ldots, s_n$ be a string. An *interchange* of elements s_i and s_j, $i < j$, transforms S into $S' = s_1, \ldots, s_{i-1}, s_j, s_{i+1}, \ldots, s_{j-1}, s_i, s_{j+1}, \ldots, s_n$.

Cost models. There are two known cost models in the context of rearrangement problems. In the *Unit-cost model (UCM)* each operation is given a unit cost, so the problem is to transform S into T with a minimum number of operations. In the *Length-cost model (LCM)*, the cost of an operation depends on its length characteristic. Other characteristics may be considered in the rearrangement problem. For example, some elements may be heavier than other elements. In such cases, moving light elements is preferable to moving heavy elements. This observation motivated researchers to explore the *Element-cost model (ECM)*. In [12], Gupta and Kumar considered the problem of sorting and selection in the comparison model for structured costs. In their work, they assumed that every element has a weight and that the cost of a comparison is defined by a function applied to the weight of the elements that participate in the comparison. They gave approximations for the optimal solution for families of structured functions such as summation, multiplication, etc. Recently, [5] addressed the same problem of sorting and selection for random costs. However, this paper is the first to consider the *ECM* for dealing with rearrangement problems.

Definition 2. Let $w : \Sigma \to \mathbb{R}^+$ be a weight function, which assigns a non-negative weight to every element in Σ. Let $g : \Sigma \times \Sigma \to \mathbb{R}^+$ be a function defining the interchange cost. The function g is called a *general function* if it satisfies the following conditions:

1. $\forall x, y \in \Sigma : g(x, y) = g(y, x)$.
2. $\forall x, y, z \in \Sigma : w(y) \leq w(z) \Leftrightarrow g(x, y) \leq g(x, z)$.

The summation function $g(x, y) = w(x) + w(y)$ and the multiplication function $g(x, y) = w(x) \cdot w(y)$ are two examples of intuitive general functions. The technique used in the *interchange rearrangement problem* under the *ECM* is different than the one used under the *UCM*. Consider the example shown in Fig. 1. In this example, an optimal rearrangement is given when the *UCM* is used – S is transformed into T using 3 interchanges (Fig. 1(a)). When the *ECM* is used, the same sequence of interchanges costs 900, whereas the alternative sequence of interchanges suggested performs 5 interchanges and costs only 850 (Fig. 1(b)).

If all elements in S are distinct, a unique bijection $f : S \to \{1, \ldots, n\}$ can be defined such that $f(s_i)$ equals the position of the element s_i in T. Thus S can be represented by $\pi = f(s_1), f(s_2), \ldots, f(s_n)$ and T by $1, \ldots, n$. For this case the term *permutation string* is used. The input string is then assumed to be π, i.e, a permutation of $1, \ldots, n$. Under this assumption the rearrangement problem is simply a sorting problem, i.e. the distance is the minimum cost for sorting π. Problems of sorting a *permutation string* have been studied extensively [6–8,10,14,15]. For the general case in which S may have repetitions of elements, the term *general string* is used.

Table 1
A summary of results.

	UCM	ECM	LCM
Interchanges			
Permutation strings	$O(n)$ [9]	$O(n)$ for a general function	$O(n)$ [4]
General strings	$\mathcal{NP}$-hard [4] $O(n \cdot \lg \lvert\Sigma\rvert)$ 1.5-approx. [4]	$\mathcal{NP}$-hard $O(n)$ 3-approx. for a general function $O(n \cdot \lg \lvert\Sigma\rvert)$ 1.72-approx. for the summation function	$O(n)$ [4]
Transpositions			
Permutation strings	$O(n \lg n)$ [14]	$O(n \lg n)$	$O(n \lg n)$
General strings	$O(n^2)$	$O(n^2)$	$O(n \lg n)$

Results. Our main results are:

1. $O(n)$ time algorithm for the *interchange rearrangement problem* for *permutation strings* for any general function.
2. $\mathcal{NP}$-hardness for the *interchange rearrangement problem* for *general strings*:
   (a) $O(n)$ time 3-approximation algorithm for any general function.
   (b) $O(n \cdot \lg |\Sigma|)$ time 1.72-approximation algorithm for the summation function.

We also broaden the study to include the *transposition rearrangement problem* under the *ECM*, *UCM* and the *LCM* for *general strings* and *permutation strings*. Table 1 summarizes the known and new results.

The paper is organized as follows. Section 2 gives additional preliminaries and notations. Section 3 presents an algorithm for the *interchange rearrangement problem* for *permutation strings* for any *general function*. Section 4 presents an approximation algorithm for the *interchange rearrangement problem* for *general strings* for any *general function* and an improved approximation algorithm for the summation function. Finally, Section 5 presents a simple extension of the *transposition rearrangement problem* under the *ECM*, *UCM* and the *LCM* in order to give an additional perspective on the new model.

2. Preliminaries and notations

Given an input string S and a target string T, we define a multi-graph $G_{S,T} = (V, E)$ (see Fig. 2) in the following way: $V = \{v \in \Sigma : v \text{ appears in } S \text{ and } T\}$ and $E = \{(t_i, s_i), 1 \leq i \leq n\}$. In other words, every distinct character has a vertex and for every index $1 \leq i \leq n$ there is an edge connecting the vertex representing t_i with the vertex of s_i, meaning that by the end of the rearrangement process, s_i will be moved and replaced by a t_i character. Since S and T have the same quantities of each element of Σ, the number of incoming edges of every vertex equals the number of its outgoing edges, which is the number of occurrences of the vertex's character in S (and hence in T). Therefore, each of the strongly connected components of $G(S, T)$ is an Eulerian directed graph and by definition can be decomposed into edge-disjoint directed cycles. If S is a *permutation string*, every vertex has exactly one incoming edge and one outgoing edge and therefore, $G_{S,T}$ can be uniquely decomposed into edge-disjoint directed cycles. This fact is not true for *general strings*. Furthermore, there might be an exponential number of ways to decompose $G_{S,T}$ into edge-disjoint directed cycles. However, once such a decomposition of $G_{S,T}$ is given, it uniquely defines a labeling of the elements of S and T such that every element appears exactly once. An edge-disjoint directed cycle in a given decomposition is also called a *permutation cycle*. A *permutation cycle* represents a subsequence of a permutation whose elements trade places cyclically. We use the following notations:

- $d(pi)$: The distance in the *permutation string* case (the minimum cost for sorting π) and $d(S, T)$ in the *general string* case (the minimum cost for transforming S into T).
- $e \leftrightarrow f$: Denotes the operation of interchanging elements e and f. Note that if e and f appear in the same cycle, interchanging them splits their cycle into two cycles. If e and f appear in different cycles, interchanging them unites their cycles into one cycle (see Fig. 2(a), (b)).
- $S_{\min}$: Denotes the minimum cost element in S. If the input string is a *permutation string* we substitute this notation with $\pi_{\min}$.
- $\tilde{S}$: Denotes the multi-set of elements that are not in place. For example, if $T = abcab$ and $S = bbaca$ then $\tilde{S} = \{a, a, b, c\}$.

The following notations apply directly to a *permutation string*. However, given a decomposition of $G_{S,T}$ into edge-disjoint directed cycles in the case of a *general string*, these notations may be also applied. We use the notation G_π instead of $G_{S,T}$ for the case of a *permutation string*:

- For a cycle C:
  - $|C|$: Denotes the number of elements in C (the size of C). We use the term ℓ-cycle for a cycle of size ℓ.
  - $C_{\min}$: Denotes the minimum cost element in C.
- $c(\pi)$: Denotes the number of cycles in G_π.

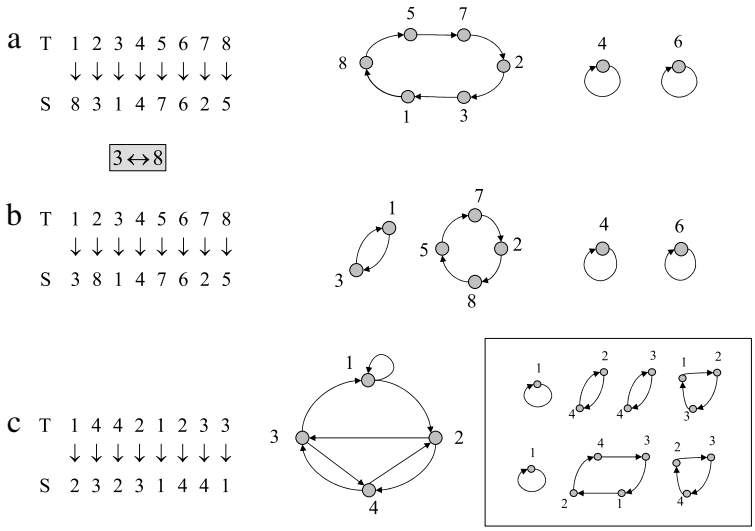

Fig. 2. In (a) and (b), S is a *permutation string*. Thus, every vertex in G_π has exactly one incoming edge and one outgoing edge and G_π is in fact the unique edge-disjoint directed cycles decomposition. Interchanging $3 \leftrightarrow 8$ in (a) splits their cycle into two cycles as shown in (b). The same interchange in (b) unites their two cycles into one cycle, as shown in (a). In (c), S is a *general string* and is a permutation of T. Therefore, the number of incoming edges equals the number of outgoing edges and equals the number of occurrences in S (or in T). Hence, $G_{S,T}$ is an Eulerian directed graph, and can be decomposed into edge-disjoint directed cycles. However, this decomposition is not unique.

3. Sorting a *permutation string*

In this section we demonstrate an algorithm for the *interchange rearrangement problem* when the input string is a *permutation string* for any general function under the *ECM*. This problem is defined as follows:

Definition 3. Let π be a *permutation string* and let $g : \Sigma \times \Sigma \to \mathbb{R}^+$ be a general function. Compute the minimum cost for sorting π by interchanges when the cost of interchanging elements x and y is defined by $g(x, y)$.

Cayley [9] studied this problem under the *UCM*. He showed that given a permutation π, the minimum number of interchanges needed for sorting π, is $n - c(\pi)$. This is achieved by interchanging only elements that share a cycle until there are no such elements (the permutation is sorted). When the *ECM* is used, one might also be inclined to apply a minimum number of interchanges. This inclination implies that one would be making interchanges only within cycles. Any interchange between elements of different cycles would result in an increase in the number of interchanges needed for sorting π and probably in the total cost for sorting π. However, this inclination is incorrect. Moreover, there might be cases in which the optimal solution would be to increase the number of interchanges needed for sorting π in order to decrease the total cost. We will describe an algorithm for sorting a *permutation string* by *interchanges* under *ECM*, and then prove that it yields the optimal cost, i.e., the distance $d(\pi)$.

3.1. The $O(n)$ time algorithm

The basic idea of the CEA_{ps} algorithm (Fig. 4) is quite simple. In order to sort the permutation π at a minimum cost, either the cheapest element in some cycle is used to sort all the other elements including itself, or (if the cheapest element in the cycle is not cheap enough) the cost for introducing the cycle to the cheapest element in π is "paid" by interchanging it with the cheapest element of the cycle. Doing so unites the cycle with the cycle of the minimum cost element of π. Then the cheapest element of π can be used to sort all the other elements in the cycle. We call this algorithm "*The Cheapest Employee Algorithm" (CEA)*.

Definition 4. Let C be a cycle in G_π, define:

- $\alpha_{in}(C) = \sum_{x \in C \setminus \{C_{\min}\}} g(C_{\min}, x) = \sum_{x \in C} g(C_{\min}, x) - g(C_{\min}, C_{\min})$

This represents the case in which a cycle C is sorted within itself, i.e. by using only interchanges of elements within C. This is done by repeatedly interchanging $C_{\min}$ with the other elements in C as shown in Fig. 3(a) until all C's elements including $C_{\min}$ are sorted.

- $\alpha_{out}(C) = \sum_{x \in C} g(\pi_{\min}, x) + g(\pi_{\min}, C_{\min})$

This represents the case in which in order to sort the elements of C, $\pi_{\min}$ is introduced to C by interchanging $C_{\min}$ with $\pi_{\min}$. The result of this interchange is that the elements of C in the new united cycle form a connected path and $\pi_{\min}$ is

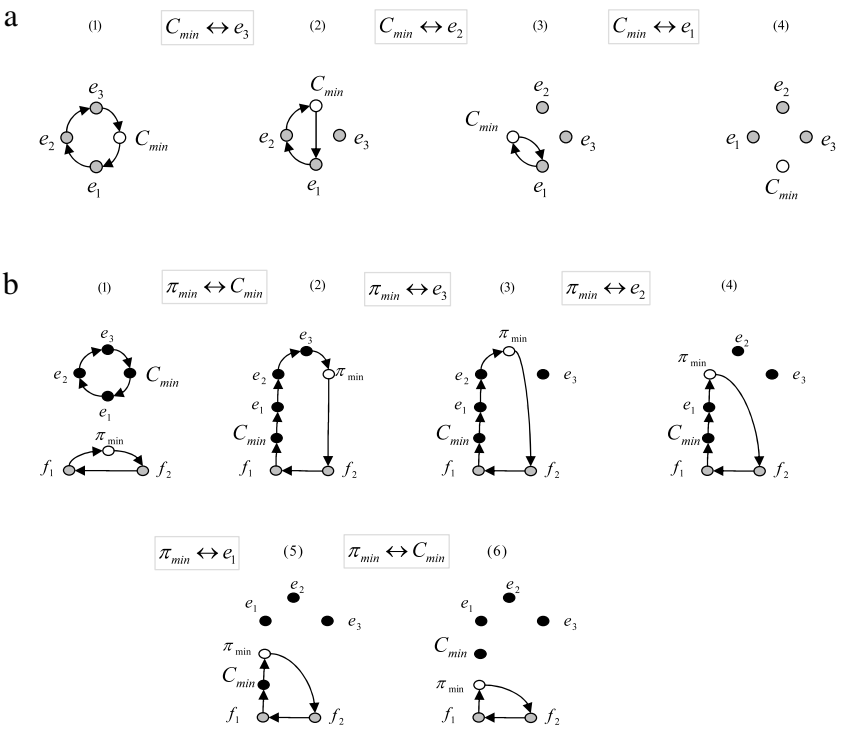

Fig. 3. In (a) the sorting is done within the cycle using its minimum cost element, $C_{\min}$. In (b) the sorting is done by introducing the cycle to the minimum cost element, $\pi_{\min}$. Note that after the interchange $C_{\min} \leftrightarrow \pi_{\min}$ the elements of C form a connected path in the new cycle (the black vertices path) and $\pi_{\min}$ is positioned at the tail of this path (white vertex).

CEA_{ps} algorithm

Data : A *permutation string* π, a general function $g : \Sigma \times \Sigma \to \mathbb{R}^+$
Result : Sorts π and returns the cost
begin

1. Compute $C_1, \ldots, C_{c(\pi)}$ and $\pi_{min}, C_{1_{min}}, \ldots, C_{c(\pi)_{min}}$
2. cost$\leftarrow 0$
3. For $1 \le i \le c(\pi)$ do
4. Compute $\alpha_{in}(C_i)$ and $\alpha_{out}(C_i)$
5. If $\alpha_{in}(C_i) \le \alpha_{out}(C_i)$
6. While $\exists e \in C_i$ with an edge $(e, C_{i_{min}})$ and $|C_i| \neq 1$ do
7. $C_{i_{min}} \leftrightarrow e$
8. cost $\leftarrow$ cost$+\alpha_{in}(C_i)$
9. Else
10. $C_{i_{min}} \leftrightarrow \pi_{min}$
11. While $\exists e \in C_i$ with an edge (e, π_{min}) do
12. $\pi_{min} \leftrightarrow e$
13. cost$\leftarrow$ cost$+\alpha_{out}(C_i)$
14. return *cost*

end

Fig. 4. Algorithm for sorting a *permutation string* by *interchanges* under *ECM*.

positioned at the tail of this path. Then $\pi_{\min}$ is interchanged with all the elements of C in order to sort them in the same manner described for $\alpha_{in}(C)$ (see Fig. 3(b)).

- $\alpha(C) = \min\{\alpha_{in}(C), \alpha_{out}(C)\}$
  The minimum cost method for sorting C.

Step 1 of the CEA_{ps} algorithm (Fig. 4) computes the permutation cycles of π. This is done by a left to right traversal of π. In addition, the minimum cost element for every cycle and for the whole *permutation string* is computed. Then, in steps 3–13, each cycle is tested separately for the cheapest sorting method and this method is applied.

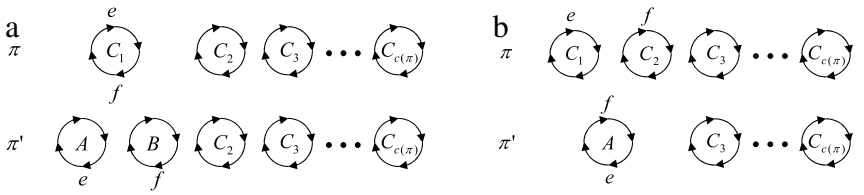

Fig. 5. In case 1 − (a), $e, f \in C_1$ and after the interchange $e \leftrightarrow f$: $e \in A$ and $f \in B$. In case 2 − (b), $e \in C_1$ and $f \in C_2$ and after the interchange $e \leftrightarrow f$: $e, f \in A$.

3.2. Correctness of the algorithm

In this subsection we show that the CEA_{ps} algorithm is optimal, i.e, returns the distance $d(\pi)$. The cost returned by the CEA_{ps} algorithm defines an upper bound for the distance, which is:

$$d(\pi) \leq \sum_{1 \leq i \leq c(\pi)} \alpha(C_i)$$

We now show that it matches the lower bound.

Lemma 1. *Let π be a* permutation string *and let $C_1, \ldots, C_{c(\pi)}$ be the cycles of G_π, then:*

$$d(\pi) \geq \sum_{1 \leq i \leq c(\pi)} \alpha(C_i)$$

Proof. By induction on the number of interchanges performed by the optimal solution. The case in which the optimal solution performs 0 operations is trivial (a sorted permutation). Assume that the lemma applies for a permutation that can be optimally sorted in $k-1$ interchanges. We prove that the lemma also applies for a permutation that can be optimally sorted in k interchanges. Let π be a permutation of $1, \ldots, n$ with cycles $C_1, \ldots, C_{c(\pi)}$, which can be optimally sorted in k interchanges. Suppose that the first interchange of this solution is $e \leftrightarrow f$. Then the resulting permutation after performing this interchange is a permutation π', which can be optimally sorted in $k-1$ operations. Thus π' satisfies the induction hypothesis. The cost for sorting π is: $d(\pi) = d(\pi') + g(e, f)$. There are two cases to consider:

Case 1: e and f in π belong to the same cycle. Assume w.l.o.g. that $e, f \in C_1$ and after performing the interchange, $e \in A$ and $f \in B$ (see Fig. 5(a)). The distance is:

$$d(\pi) = d(\pi') + g(e, f) \geq \alpha(A) + \alpha(B) + \sum_{2 \leq i \leq c(\pi)} \alpha(C_i) + g(e, f)$$

In order to prove the lemma for this case, we need to show that $\alpha(A) + \alpha(B) + g(e, f) \geq \alpha(C_1)$. We use the following simple arguments:

1. $w(\pi_{\min}) \leq w(C_{1_{\min}}) \leq \begin{array}{ll} w(A_{\min}) \leq w(x), & \forall x \in A \\ w(B_{\min}) \leq w(x), & \forall x \in B \end{array}$
2. $A \cup B = C_1, A \cap B = \emptyset$

There are three subcases to consider:

Case 1.1: $\alpha(A) = \alpha_{in}(A)$ and $\alpha(B) = \alpha_{in}(B)$. If both A and B are sorted within themselves then obviously C_1 is sorted using only interchanges inside C_1. Since either A or B might be a cycle with a minimum cost element that is more expensive than $C_{1_{\min}}$, the cost for sorting A and B in addition to the interchange of elements e and f might be more expensive, but never cheaper than sorting C_1 within. Assume w.l.o.g. that $A_{\min} = C_{1_{\min}}$. Thus:

$$\begin{aligned} \alpha_{in}(A) + \alpha_{in}(B) + g(e, f) &= \sum_{x \in A \setminus \{A_{\min}\}} g(A_{\min}, x) + \sum_{x \in B \setminus \{B_{\min}\}} g(B_{\min}, x) + g(e, f) \\ &\geq \sum_{x \in C_1 \setminus \{A_{\min}, B_{\min}\}} g(C_{1_{\min}}, x) + g(C_{1_{\min}}, B_{\min}\}) \\ &= \sum_{x \in C_1 \setminus \{C_{1_{\min}}\}} g(C_{1_{\min}}, x) \\ &= \alpha_{in}(C_1) \geq \alpha(C_1) \end{aligned}$$

Case 1.2: W.l.o.g. $\alpha(A) = \alpha_{in}(A)$ and $\alpha(B) = \alpha_{out}(B)$. This case implies that the extra cost for introducing B to $\pi_{\min}$ is being paid. Introducing C_1 to $\pi_{\min}$ will result in a cheaper cost because A may also benefit from it. Thus:

$$\begin{aligned} \alpha_{in}(A) + \alpha_{out}(B) + g(e, f) &= \sum_{x \in A \setminus \{A_{\min}\}} g(A_{\min}, x) + \sum_{x \in B} g(\pi_{\min}, x) + g(\pi_{\min}, B_{\min}) + g(e, f) \\ &\geq \sum_{x \in C_1 \setminus \{A_{\min}\}} g(\pi_{\min}, x) + g(\pi_{\min}, C_{1_{\min}}) + g(\pi_{\min}, A_{\min}) \\ &= \alpha_{out}(C_1) \geq \alpha(C_1) \end{aligned}$$

Case 1.3: $\alpha(A) = \alpha_{out}(A)$ **and** $\alpha(B) = \alpha_{out}(B)$**.** This case implies that an extra cost is paid for both A and B for introducing them to $\pi_{\min}$. Instead of paying that extra cost for two cycles, it would be cheaper to pay this extra cost only once for one cycle. Thus:

$$\begin{aligned}\alpha_{out}(A)+\alpha_{out}(B)+g(e,f) &= \sum_{x\in A} g(\pi_{\min},x)+g(\pi_{\min},A_{\min})+\sum_{x\in B} g(\pi_{\min},x)+g(\pi_{\min},B_{\min})+g(e,f)\\ &\geq \sum_{x\in C_1} g(\pi_{\min},x)+g(\pi_{\min},C_{1_{\min}})\\ &= \alpha_{out}(C_1)\geq\alpha(C_1)\end{aligned}$$

Case 2: e **and** f **in** π **belong to different cycles.** Assume w.l.o.g. that $e\in C_1$ and $f\in C_2$ and after performing the interchange $e,f\in A$ (see Fig. 5(b)). The distance is:

$$d(\pi)=d(\pi')+g(e,f)\geq\alpha(A)+\sum_{3\leq i\leq c(\pi)}\alpha(C_i)+g(e,f)$$

In order to prove the lemma for this case, we need to show that $\alpha(A)+g(e,f)\geq\alpha(C_1)+\alpha(C_2)$. In the two subcases below we assume w.l.o.g. that $A_{\min}=C_{1_{\min}}$ and we use the following simple arguments:

1. $w(\pi_{\min})\leq w(A_{\min})$ $\begin{array}{l}=w(C_{1_{\min}})\leq w(x),\ \forall x\in C_1\\ \leq w(C_{2_{\min}})\leq w(x),\ \forall x\in C_2\end{array}$
2. $C_1\cup C_2=A,\ C_1\cap C_2=\emptyset$

There are two subcases to consider:
Case 2.1: $\alpha(A)=\alpha_{in}(A)$**.** This case implies that A is being sorted within itself. The only cycle that may benefit from the union is C_2, because its minimum cost element, $C_{2_{\min}}$, might be more expensive than $C_{1_{\min}}=A_{\min}$. Since $C_{1_{\min}}$ may be more expensive than $\pi_{\min}$, C_2 may benefit more from uniting with the cycle of $\pi_{\min}$. Thus:

$$\begin{aligned}\alpha_{in}(A)+g(e,f) &= \sum_{x\in A\setminus\{A_{\min}\}} g(A_{\min},x)+g(e,f)\\ &\geq \sum_{x\in C_1\setminus\{C_{1_{\min}}\}} g(C_{1_{\min}},x)+\sum_{x\in C_2} g(\pi_{\min},x)+g(\pi_{\min},C_{2_{\min}})\\ &= \alpha_{in}(C_1)+\alpha_{out}(C_2)\geq\alpha(C_1)+\alpha(C_2)\end{aligned}$$

Case 2.2: $\alpha(A)=\alpha_{out}(A)$**.** This case implies that the extra cost for introducing A to $\pi_{\min}$ is being paid. There are two interchanges performed here, which result in uniting the cycles C_1 and C_2 with the cycle of $\pi_{\min}$. These two operations cost us exactly $g(A_{\min},\pi_{\min})+g(e,f)$. However, the same result can be achieved with perhaps a cheaper cost (but never more expensive). Simply unite C_1 with the cycle of $\pi_{\min}$ and C_2 with the cycle of $\pi_{\min}$ separately. This will cost $g(C_{1_{\min}},\pi_{\min})+g(C_{2_{\min}},\pi_{\min})$ and may only be cheaper. Thus:

$$\begin{aligned}\alpha_{out}(A)+g(e,f) &= \sum_{x\in A} g(\pi_{\min},x)+g(\pi_{\min},A_{\min})+g(e,f)\\ &\geq \sum_{x\in C_1} g(\pi_{\min},x)+\sum_{x\in C_2} g(\pi_{\min},x)+g(\pi_{\min},C_{1_{\min}})+g(\pi_{\min},C_{2_{\min}})\\ &= \alpha_{out}(C_1)+\alpha_{out}(C_2)\geq\alpha(C_1)+\alpha(C_2).\quad\square\end{aligned}$$

Theorem 1 immediately follows from the upper bound of the algorithm and Lemma 1.

Theorem 1. *Let* π *be a* permutation string *and let* $C_1,\ldots,C_{c(\pi)}$ *be the cycles of* G_π*. Then the minimum cost for sorting* π *by* interchanges *under* ECM *for any general function is:*

$$d(\pi)=\sum_{1\leq i\leq c(\pi)}\alpha(C_i).$$

Complexity: By Theorem 1, the CEA_{ps} algorithm computes the distance $d(\pi)$. Computing the permutation cycles can be done in linear time by a left to right traversal. Also, testing all the cycles is done in linear time, since the first element e in the adjacency list of $C_{i_{\min}}$ (or $\pi_{\min}$) can be taken in $O(1)$ time. The interchange $(e,C_{i_{\min}})$ (resp. $(e,\pi_{\min})$) then sorts e and the original cycle is shortened by one element, but $C_{i_{\min}}$ (or $\pi_{\min}$) are still in the cycle, so this process can be repeated until all elements in the cycle are sorted. Therefore, the CEA_{ps} algorithm runs in linear time.

CEA_{gs} algorithm.

Data : Input string S, target string T, a general function $g : \Sigma \times \Sigma \to \mathbb{R}^+$
Result : Transform S into T
begin

1. Compute $G_{S,T}$.
2. Compute a decomposition D of $G_{S,T}$ as follows:
3. $D \leftarrow \emptyset$.
4. Add to D all the 1-cycles of $G_{S,T}$ and remove their edges.
5. Add to D an arbitrary decomposition of the remaining edges.
6. Apply the CEA_{ps} algorithm on D.

end

Fig. 6. 3-approximation algorithm for the *interchange rearrangement problem* under *ECM* for *general strings* for a general function g.

4. Rearranging *general strings*

In the previous section we showed a linear time algorithm that computes the distance in the *interchange rearrangement problem* when the input string is a *permutation string* and for every general function. In this section we consider the following problem:

Definition 5. Let S be the input string and T be the target string, when S is a permutation of T and let $g : \Sigma \times \Sigma \to \mathbb{R}^+$ be a general function. Compute the minimum cost for transforming S into T by interchanges when the cost of interchanging elements x and y is defined by $g(x, y)$.

The *interchange rearrangement problem* under the *UCM* for *general strings* is $\mathcal{NP}$-hard [4]. Hence, as the *UCM* is a special case of *ECM* where all elements have equal weights, Corollary 1 follows:

Corollary 1. *The* interchange rearrangement problem *under* ECM *for* general strings *is $\mathcal{NP}$-hard.*

In the following subsection we present an $O(n)$ time, 3-approximation algorithm for any general function. In addition, we present an $O(n \cdot \lg|\Sigma|)$ time 1.72-approximation algorithm for the summation function.

4.1. $O(n)$ time 3-approximation algorithm for general functions

The hardness of this problem is due to the difficulty of pairing each element in S with an identical element in T (converting the problem into a *permutation string* problem) in a way that gives the minimum distance. As explained in Section 2, pairing elements from S with elements in T is equivalent to performing an edge-disjoint decomposition of $G_{S,T}$ into directed cycles. Since S is a permutation of T, each of the strongly connected components the graph graph $G_{S,T}$ is an Eulerian directed graph and such a decomposition exists. The CEA_{gs} algorithm (Fig. 6) arbitrarily decomposes $G_{S,T}$ into cycles and then applies the CEA_{ps} algorithm (Fig. 4). We prove the following theorem:

Theorem 2. *The CEA_{gs} algorithm gives a 3-approximation ratio for any general function.*

Proof. We first observe that any solution for the problem implies a decomposition of $G_{S,T}$ into edge-disjoint directed cycles. This is true because any solution implies a pairing of the elements of S and T, which is equivalent to performing such a decomposition. Assume that the optimal solution implies a decomposition of $G_{S,T}$ into cycles $C_1, \ldots, C_k$. Then by Theorem 1:

$$\begin{aligned} d(S,T) &= \sum_{i=1}^{k} \alpha(C_i) \\ &= \sum_{i=1}^{k} \min\left\{ \sum_{x\in C_i} g(C_{i_{\min}}, x) - g(C_{i_{\min}}, C_{i_{\min}}) \,,\; \sum_{x\in C_i} g(S_{\min}, x) + g(C_{i_{\min}}, S_{\min}) \right\} \end{aligned}$$

Since $w(S_{\min}) \le w(C_{i_{\min}})$ then by decreasing the weight of $C_{i_{\min}}$, $\forall 1 \le i \le k$ to $w(S_{\min})$ the total cost may only decrease:

$$d(S,T) \ge \sum_{i=1}^{k} \left(\sum_{x\in C_i} g(S_{\min}, x) - g(S_{\min}, C_{i_{\min}}) \right)$$

Define $Z = \sum_{x\in\tilde{S}} g(S_{\min}, x) = \sum_{i=1}^{k} \sum_{x\in C_i} g(S_{\min}, x)$. The expression $\sum_{i=1}^{k} g(S_{\min}, C_{i_{\min}})$ is bounded by the case when all cycles are 2-cycles. Since for every 2-cycle, C, with elements x and $C_{\min}$: $g(S_{\min}, C_{\min}) \le \frac{1}{2}(g(S_{\min}, x) + g(S_{\min}, C_{\min}))$, it follows that $\sum_{i=1}^{k} g(S_{\min}, C_{i_{\min}}) \le \frac{1}{2}Z$. Therefore, a lower bound for the distance of the optimal solution is:

$$d(S,T) \ge Z - \frac{1}{2}Z = \frac{1}{2}Z$$

We now show an upper bound on the distance computed by the CEA_{gs} algorithm, denoted by d_{alg}. Consider a modified version of the CEA_{gs} algorithm that sorts each cycle in the decomposition D with the α_{out} sorting method. Since the CEA_{ps} applied in step 6 of the CEA_{gs} is optimal, the distance computed by the CEA_{gs} algorithm may only be lower than the distance computed by the modified version. Let $C_1, \ldots, C_l$ be the cycles arbitrarily decomposed by the CEA_{gs} algorithm. We therefore have:

$$\begin{aligned} d_{alg} &\leq \sum_{i=1}^{l} \left(\sum_{x \in C_i} g(S_{\min}, x) + g(S_{\min}, C_{i_{\min}}) \right) \\ &\leq Z + \frac{1}{2}Z = 1\frac{1}{2}Z \end{aligned}$$

The ratio between d_{alg} and $d(S,T)$ is: $\frac{d_{alg}}{d(S,T)} \leq \frac{1\frac{1}{2}Z}{\frac{1}{2}Z} = 3$. □

Complexity: Since a $G_{S,T}$ computation and an arbitrary decomposition of $G_{S,T}$ can be computed in linear time and since the CEA_{ps} algorithm is a linear time algorithm, the CEA_{gs} algorithm runs in linear time.

4.2. $O(n \cdot \lg|\Sigma|)$ time 1.72-approximation algorithm for the summation function

In this subsection we consider the special case of the summation function, i.e., $g(x,y) = w(x) + w(y)$. The $\alpha_{in}(C)$, $\alpha_{out}(C)$ for a given cycle are therefore defined as follows:

- $\alpha_{in}(C) = \sum_{x \in C \setminus \{C_{\min}\}} g(C_{\min}, x) = \sum_{x \in C} w(x) + (|C| - 2) \cdot w(C_{\min})$
- $\alpha_{out}(C) = \sum_{x \in C} g(S_{\min}, x) + g(S_{\min}, C_{\min}) = \sum_{x \in C} w(x) + (|C| + 1) \cdot w(S_{\min}) + w(C_{\min})$

We show that applying the CEA_{ps} algorithm on the decomposition presented by [4] gives a 1.72-approximation ratio. The decomposition presented by [4] is basically the same as the decomposition of the CAE_{gs} except that it contains a maximum number of 2-cycles. This difference is represented by step 5 of the CAE^{+}_{gs} (Fig. 7). We use the following lemma, proved by [4]:

Lemma 2 (*[4]*). *Given an Eulerian directed graph $G = (V, E)$, then for every 2-cycle, C, in G there exists a decomposition of E into a maximum number of edge-disjoint directed cycles, in which C appears as a cycle in the decomposition.*

By Lemma 2 there exists a decomposition of $G_{S,T}$ into a maximum number of edge-disjoint directed cycles that contains a maximum number of 2-cycles. This can be shown inductively by repeatedly finding a 2-cycle and removing it from $G_{S,T}$ until there are no more 2-cycles. By Lemma 2 in every stage, there exists a decomposition into a maximum number of edge-disjoint directed cycles that contains the chosen 2-cycle. Removing it results in a new graph G', for which all its strongly connected components are Eulerian directed graphs. Therefore, the same can be applied for G'. We prove the following theorem:

Theorem 3. *The CEA^{+}_{gs} algorithm gives a 1.72-approximation ratio.*

Proof. We begin by giving a lower bound for the distance. Denote by $\#c_2$ the maximum number of 2-cycles that can be decomposed from $G_{S,T}$, by m the number of edges that remain after removing all the 1-cycles and a maximum number of 2-cycles from $G_{S,T}$. Clearly $|\tilde{S}| = m + 2 \cdot \#c_2$. Since the maximal number of cycles that can be decomposed from the remaining m elements is $\frac{m}{3}$ (when the remaining m elements are decomposed into 3-cycles), $\#c_2 + \frac{m}{3}$ is an upper bound for the maximum number of edge-disjoint directed cycles of any decomposition of $G_{S,T}$. Assume that the optimal algorithm implies a decomposition of $G_{S,T}$ into cycles $C^o_1, \ldots, C^o_k$ ($k \leq \#c_2 + \frac{m}{3}$). Thus, by Theorem 1:

$$\begin{aligned} d(S,T) &= \sum_{1 \leq i \leq k} \alpha(C^o_i) \\ &= \sum_{x \in \tilde{S}} w(x) + \sum_{1 \leq i \leq k} \min\left\{ w(C^o_{i_{\min}}) \cdot (|C^o_i| - 2) \;,\; w(S_{\min}) \cdot (|C^o_i| + 1) + w(C^o_{i_{\min}}) \right\} \\ &\geq \sum_{x \in \tilde{S}} w(x) + \sum_{1 \leq i \leq k} (w(S_{\min}) \cdot (|C^o_i| - 2)) \end{aligned}$$

Note that the cost for every 2-cycle is exactly the sum of the cost of its two elements since for a 2-cycle C^o_i, $|C^o_i| - 2 = 0$. Assume w.l.o.g. that the l last cycles are 2-cycles. The number of elements in the remaining $k - l$ cycles is exactly $|\tilde{S}| - 2l$. Thus:

$$\begin{aligned} d(S,T) &\geq \sum_{x \in \tilde{S}} w(x) + w(S_{\min}) \cdot \left(\sum_{1 \leq i \leq k-l} |C^o_i| - 2(k-l) \right) \\ &= \sum_{x \in \tilde{S}} w(x) + w(S_{\min}) \cdot (|\tilde{S}| - 2k) \end{aligned}$$

CEA^{+}_{gs} algorithm.

Data : Input string S, target string T
Result : Transform S into T
begin

1. Compute $G_{S,T}$.
2. Compute a decomposition D of $G_{S,T}$ as follows:
3. $D \leftarrow \emptyset$.
4. Add to D all the 1-cycles of $G_{S,T}$ and remove their edges.
5. Add to D a maximum number of 2-cycles from $G_{S,T}$ and remove their edges.
6. Add to D an arbitrary decomposition of the remaining edges.
7. Apply the CEA_{ps} algorithm on D.

end

Fig. 7. 1.72-Approximation algorithm for the *interchange rearrangement problem* under *ECM* for *general strings* for the summation function.

As $|\tilde{S}| = m + 2 \cdot \#c_2$ and since $k \leq \#c_2 + \frac{m}{3}$ then:

$$\begin{aligned} d(S,T) &\geq \sum_{x\in\tilde{S}} w(x) + w(S_{\min}) \cdot \left(m + 2 \cdot \#c_2 - 2\left(\#c_2 + \frac{m}{3}\right)\right) \\ &= \sum_{x\in\tilde{S}} w(x) + \frac{m \cdot w(S_{\min})}{3} \end{aligned}$$

We now prove the 1.72-approximation ratio of the CEA^{+}_{gs} algorithm (Fig. 7). Consider a modified version of the CEA^{+}_{gs} algorithm that instead of step 7, which applies the CAE_{ps} algorithm, sorts small cycles (cycles of size 3–7) with the α_{in} sorting method and large cycles (cycles of size greater than 7) with the α_{out} sorting method. As the CEA_{ps} algorithm is optimal, the cost of the CEA^{+}_{gs} algorithm may only be lower than the cost of the modified version. Denote the number of small cycles by $\#c_7$ and the number of large cycles by $\#c_8$. Denote the set of all elements that belong to 2-cycles, small cycles and large cycles by C_2, C_7, C_8 respectively. Denote the number of elements that belong to small cycles and large cycles by m_7, m_8 respectively. Note that $m = m_7 + m_8$ and that $\sum_{x\in\tilde{S}} w(x) = \sum_{x\in C_2} w(x) + \sum_{x\in C_7} w(x) + \sum_{x\in C_8} w(x)$. The lower bound of the problem can be rewritten as:

$$\sum_{x\in\tilde{S}} w(x) + \frac{m \cdot w(S_{\min})}{3} = \underbrace{\sum_{x\in C_2} w(x)}_{a} + \underbrace{\sum_{x\in C_7} w(x) + \frac{m_7}{3} \cdot w(S_{\min})}_{b} + \underbrace{\sum_{x\in C_8} w(x) + \frac{m_8}{3} \cdot w(S_{\min})}_{c}$$

The CEA^{+}_{gs} algorithm pays exactly the cost for invariant a. We now analyze the cost for invariants b and c. We use the following arguments:

1. For a cycle C_i: $w(C_{i_{\min}}) \leq \frac{\sum_{x\in C_i} w(x)}{|C_i|}$
2. $\#c_8 \leq \frac{m_8}{8}$
3. $\sum_{x\in C_8} w(x) \geq m_8 \cdot w(S_{\min})$
4. $\forall x, y, z \geq 0$: $\frac{x+y}{x+z} \geq 1$ and $0 \leq x' \leq x \Rightarrow \frac{x+y}{x+z} \leq \frac{x'+y}{x'+z}$

Denote by b_{alg} the cost for sorting all the small cycles (using α_{in}). Assume that the small cycles are $C^s_1, \ldots, C^s_{\#c_7}$ Using argument 1, b_{alg} is bounded by:

$$b_{alg} = \sum_{x\in C_7} w(x) + \sum_{1\leq i\leq \#c_7} w(C^s_{i_{\min}}) \cdot |C^s_i| - 2 \leq 1\frac{5}{7} \sum_{x\in C_7} w(x)$$

The ratio between b_{alg} and invariant b is:

$$\frac{b_{alg}}{b} \leq \frac{1\frac{5}{7} \sum_{x\in C_7} w(x)}{\sum_{x\in C_7} w(x) + \frac{m_7}{3} \cdot w(S_{\min})} \leq 1\frac{5}{7} \leq 1.72$$

Denote by c_{alg} the cost for sorting all the large cycles (using α_{out}). Assume that the large cycles are $C^l_1, \ldots, C^l_{\#c_8}$. Using arguments 1, 2, 3, c_{alg} is bounded by:

$$\begin{aligned} c_{alg} &= \sum_{x\in C_8} w(x) + w(S_{\min}) \cdot \sum_{1\leq i\leq \#c_8} (|C^l_i| + 1) + \sum_{1\leq i\leq \#c_8} w(C^l_{i_{\min}}) \\ &= \sum_{x\in C_8} w(x) + w(S_{\min}) \cdot m_8 + w(S_{\min}) \cdot \#c_8 + \sum_{1\leq i\leq \#c_8} w(C^l_{i_{\min}}) \\ &\leq 1\frac{1}{8} \sum_{x\in C_8} w(x) + 1\frac{1}{8} m_8 \cdot w(S_{\min}) \end{aligned}$$

Using arguments 3 and 4, the ratio between c_{alg} and c is:

$$\frac{c_{alg}}{c} \le \frac{1\frac{1}{8}\sum_{x\in C_8} w(x) + 1\frac{1}{8}m_8 \cdot w(S_{\min})}{\sum_{x\in C_8} w(x) + \frac{m_8}{3} \cdot w(S_{\min})} = \frac{\frac{1}{8}\sum_{x\in C_8} w(x)}{\sum_{x\in C_8} w(x) + \frac{m_8}{3} \cdot w(S_{\min})} + \frac{\sum_{x\in C_8} w(x) + 1\frac{1}{8}m_8 \cdot w(S_{\min})}{\sum_{x\in C_8} w(x) + \frac{m_8}{3} \cdot w(S_{\min})}$$
$$\le \frac{1}{8} + \frac{m_8 \cdot w(S_{\min}) + 1\frac{1}{8}m_8 \cdot w(S_{\min})}{m_8 \cdot w(S_{\min}) + \frac{m_8}{3} \cdot w(S_{\min})} \le \frac{1}{8} + \frac{2\frac{1}{8}}{1\frac{1}{3}} \le 1.72$$

Therefore,

$$d_{alg} \le a_{alg} + b_{alg} + c_{alg} \le 1.72 \cdot (a + b + c) \le 1.72 \cdot d(S, T). \quad \square$$

Complexity: The CEA^{+}_{gs} algorithm differs from the CEA_{gs} algorithm only in step 5 of CEA^{+}_{gs}. Finding a maximum number of 2-cycles in $G_{S,T}$ can be done in $O(n \cdot \lg(|\Sigma|))$ time in the following way. For each edge (of total n edges) in the graph check if there exists an edge in the opposite direction. Since there are $|\Sigma|$ nodes and the nodes can be kept ordered in the adjacency lists, this check can be done in $O(\log|\Sigma|)$ time for each edge. As a corollary of Lemma 2 repeatedly finding and removing 2-cycles this way gives a maximum number of 2-cycles. Therefore, the CEA^{+}_{gs} algorithm runs in $O(n \cdot \lg(|\Sigma|))$ time.

5. The transposition rearrangement problem

In this section we briefly discuss the *transposition rearrangement problem* in order to have a broadened view on the cost models. We refer to a single element transposition and not to a block transposition as referred to in [6,11]. We define the transposition operator as follows:

Definition 6. Let $S = s_1, \ldots, s_n$ be a string. A *transposition* of an element s_i, ℓ positions forward transforms the string S into the string $S' = s_1, \ldots, s_{i-1}, s_{i+1}, \ldots, s_{i+\ell}, s_i, s_{i+\ell+1}, \ldots, s_n$ and a *transposition* of an element s_i, ℓ positions backward transforms the string S into the string $S' = s_1, \ldots, s_{i-\ell-1}, s_i, s_{i-\ell}, \ldots, s_{i-1}, s_{i+1}, \ldots, s_n$.

Section 5.1 considers the problem under the *UCM* and under the *ECM* for both *permutation strings* and *general strings*. Section 5.2 considers the problem under the *LCM*.

5.1. Element-cost and unit-cost models

In this subsection the following problem is discussed:

Definition 7. Let S be the input string and T be the target string and let $w : \Sigma \to \mathbb{R}^+$ be a weight function. Compute the minimum cost for transforming S into T by transpositions when the cost of transposing an element x is defined by $w(x)$.

This definition generalizes all the sub-problems presented in Table 2. If S is a *permutation string*, π, the problem is to sort π at minimum cost. If $\forall x, y \in \Sigma,\ w(x) = w(y)$, the problem is to transform S into T with a minimum number of transpositions, i.e., *UCM*. For this set of problems, we use the following lemma:

Lemma 3. *In the* transposition rearrangement problem *under* UCM *or under* ECM*, each element is transposed at most once.*

Proof. Assume to the contrary that there exists an optimal solution *OPT*, such that $d_{OPT} = d(S, T)$ and *OPT* transposes an element $x\,(w(x) > 0)$ more than once. Consider the solution *OPT*′ that applies all *OPT* transpositions except for those applied on x and finally transposes x once to its position. Therefore, $d_{OPT'} < d_{OPT}$ in contradiction to the minimality of d_{OPT}. $\square$

Lemma 3 implies that in the optimal solution for the problems defined in this subsection the elements of S are divided into two sets: the set $\mathcal{A}$ of elements that are transposed exactly once and the set $\mathcal{B}$ of elements that are not transposed at all. Therefore, the distance is defined as follows:

$$d(S, T) = \sum_{x\in\mathcal{A}} w(x) = \sum_{x\in S} w(x) - \sum_{x\in\mathcal{B}} w(x)$$

Since $\mathcal{B}$ contains elements that are not transposed at all, these elements construct a common subsequence of S and T. Since $d(S, T)$ is minimized when $\sum_{x\in\mathcal{B}} w(x)$ is maximized, $\mathcal{B}$ is a common subsequence of highest cost. The details for the various problems are presented in Table 2. The Measure column indicates the relevant subsequence of the specific problem.

5.2. Length-cost model

In the *interchange rearrangement problem* under the *LCM* presented in [4], the cost of every operation was defined by applying a length function to the interchange length. They considered the $f(\ell) = \ell^{\alpha}$ length functions for every α. In this section we discuss only the case where $\alpha = 1$ (the case where $\alpha > 1$ implies only transpositions of size 1 as shown below for $\alpha = 1$, and is, therefore, the same). We consider the following problem:

Definition 8. Let S be the input string and T be the target string. Compute the minimum cost for transforming S into T by transpositions when the cost of transposing an element ℓ positions is ℓ.

Table 2
Transposition rearrangement problem under UCM and ECM.

Cost model	String type	Measure	Description	Distance[a]	Time complexity
UCM	Permutation string	*LIS*	Longest Increasing Subsequence	$n - LIS(\pi)$ [14]	$O(n \lg n)$
	General string	*LCS*	Longest Common Subsequence	$n - LCS(S, T)$	$O(n^2)$
ECM	Permutation string	*MWIS*	Maximum Weighted Increasing Subsequence	$\sum_{i=1}^{n} w(\pi_i) - MWIS(\pi)$	$O(n \lg n)$
	General string	*MWCS*	Maximum Weighted Common Subsequence	$\sum_{i=1}^{n} w(s_i) - MWCS(S, T)$	$O(n^2)$

[a] *LIS* and *LCS* refer to the size of the subsequence. *MWIS* and *MWCS* refer to the sum of the weights of the subsequence's elements.

Permutation strings

In this case, the input is a *permutation string* π and the problem is to sort π at a minimum cost. Given a permutation string π, we say that π_i and π_j are *reversed* iff $i < j$ and $\pi_i > \pi_j$. Let R^π be the set of pairs $\{i, j\}$, such that π_i and π_j are *reversed*. For example, in the string: $S = D, A, C, B$, we have $R^\pi = \{\{1, 2\}, \{1, 3\}, \{1, 4\}, \{3, 4\}\}$.

Lemma 4. *Let π be a permutation string. Then the cost for sorting π by* transpositions *under LCM is $d(\pi) = |R^\pi|$.*

Proof. A lower bound of the distance is $|R^\pi|$, since for every *reversed* pair, $\{i, j\}$, 1-length unit must be paid (either π_i must "jump" over π_j or vice versa), $d(\pi) \geq |R^\pi|$. This bound is achieved by a simple algorithm (similar to the max sort algorithm), which transposes the maximal element to the rightmost position, then transposes the remaining elements from the maximum to the minimum, by transposing each element to the left of the previous transposed element. Since the transpositions are performed from the maximum element to the minimum element, every transposed element only "jumps" over elements that are *reversed* with it and, therefore, $d(\pi) \leq |R^\pi|$. The lemma follows. □

Complexity: Computing $|R^\pi|$ can be done in $O(n \lg n)$ time by using a balanced search tree supporting position queries.

General strings

The difficulty for a *general string* input is to pair the elements of S with the elements of T in a way that gives the minimum cost. In the *interchange rearrangement problem*, this task is $\mathcal{NP}$-hard. Here, however, an optimal pairing can be defined, as stated by Lemma 5 (which can be easily verified).

Lemma 5. *Let S be the input string and T be the target string. Let π_o be the labeling that for any $a \in \Sigma$ and k, labels the k^{th} a in S as the position of the k^{th} a in T (π_o pairs the k^{th} a in S with the k^{th} a in T). Then the cost for transforming S into T by* transpositions *under* LCM *is $d(S, T) = d(\pi_o)$.*

Complexity: Since finding the labeling described in Lemma 5 can be done in $O(n \lg n)$, the total time complexity is $O(n \lg n)$.

Acknowledgments

The second author's research was partially supported by the Israel Science Foundation grant 35/05 and the Israel-Korea Scientific Research Cooperation.

References

[1] A. Amir, Y. Aumann, G. Benson, A. Levy, O. Lipsky, E. Porat, S. Skiena, U. Vishne, Pattern matching with address errors: Rearrangement distances, in: Proc. of the 17th annual ACM–SIAM Symposium on Discrete Algorithm, SODA, 2006, pp. 1221–1229.
[2] A. Amir, Y. Aumann, P. Indyk, A. Levy, E. Porat, Efficient computations of l_1 and l_∞, in: Proc. of the 14th International Symposium on String Processing and Information Retrieval, SPIRE, 2007, pp. 39–49.
[3] A. Amir, Y. Aumann, O. Kapah, A. Levy, E. Porat, Approximate string matching with address bit errors, in: Proc. of the 19th Annual Symposium on Combinatorial Pattern Matching, CPM, 2008, pp. 118–130.
[4] A. Amir, T. Hartman, O. Kapah, A. Levy, E. Porat, On the cost of interchange rearrangement in strings in: Proc. of the 15th Annual European Symposium on Algorithms, ESA, 2007, pp. 99–110.
[5] S. Angelov, K. Kunal, A. McGregor, Sorting and selection with random costs, in: Proc. of the 8th Latin American Theoretical Informatics, LATIN, 2008.
[6] V. Bafna, P.A. Pevzner, Sorting by transpositions, SIAM Journal on Discrete Mathematics 11 (1998) 224–240.
[7] P. Berman, S. Hannenhalli, Fast sorting by reversal, in: Proc. 8th Annual Symposium on Combinatorial Pattern Matching, CPM, vol. 1075, 1996, pp. 168–185.
[8] A. Carpara, Sorting by reversals is difficult, in: Proc. 1st Annual Intl. Conf. on Research in Computational Biology, RECOMB, 1997, pp. 75–83.
[9] A. Cayley, Note on the theory of permutations, Philosophical Magazine (34) (1849) 527–529.
[10] D.A. Christie, Sorting by block-interchanges, Information Processing Letters 60 (1996) 165–169.
[11] I. Elias, T. Hartman, A 1.375-approximation algorithm for sorting by transpositions, in: Proc. of the 5th International Workshop on Algorithms in Bioinformatics, WABI, 2005, pp. 204–214.
[12] A. Gupta, A. Kumar, Sorting and selection with structured costs, in: Proc. of the 42nd Symposium on Foundations of Computer Science, 2001, pp. 416–425.
[13] D. Gusfield, Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology, Cambridge University Press, 1997.
[14] L.S. Heath, Vergara, Discrete Applied Mathematics 88 (1–3) (1998) 181–206.
[15] L.S. Heath, J.P.C. Vergara, Sorting by short swaps, Journal of Computational Biology 10 (5) (2003) 775–789.